

Linear-Time Fitting of a k -Step Function

Binay Bhattacharya

School of Computing Science, Simon Fraser University, Burnaby, Canada

Sandip Das

Advanced Computing and Microelectronics Unit, Indian Stat. Inst., Kolkata, India

Tsuneiko Kameda

School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

Given a set of n weighted points on the x - y plane, we want to find a step function consisting of k horizontal steps such that the maximum weighted vertical distance from any point to a step is minimized. Using the prune-and-search technique, we solve this problem in $O(n)$ time when k is a constant. Our approach can be applied directly or with small modifications to solve other similar problems, such as the maximum error histogram problem and the line-constrained k -center problem, in $O(n)$ time when k is a constant.

Keywords: linear-time algorithm, step function fitting, weighted points, prune and search, maximum error histogram

1. Introduction

Given an integer $k > 0$ and a set P of n weighted points in the plane, our objective is to fit a k -step function to them so that the maximum weighted vertical distance of the points to the step function is minimized. We call this problem the *k -step function problem*. It has applications in areas such as geographic information systems, digital image analysis, data mining, facility locations, and data representation (histogram), etc.

In the unweighted case, if the points are presorted, Fournier and Vigneron [8] showed that the problem can be solved in linear time using the results of [10, 11, 12]. Later they showed that the weighted version of the problem can also be solved in $O(n \log n)$ time [9], using Megiddo's parametric search technique [17]. Prior to these results, the problem had been discussed by several researchers [5, 7, 15, 16, 19].

Email addresses: binay@sfu.ca (Binay Bhattacharya), sandip.das.69@gmail.com (Sandip Das), tikokameda@gmail.com (Tsuneiko Kameda)

Guha and Shim [13] considered this problem in the context of *histogram construction*. In database research, it is known as the *maximum error histogram* problem. In the weighted case, this problem is to partition the given points into k buckets based on their x -coordinates, such that the maximum y -spread in each bucket is minimized. This problem is of interest to the data mining community as well (see [13] for references). Guha and Shim [13] computed the optimum histogram of size k , minimizing the maximum error. They present algorithms which run in linear time when the points are unweighted, and in $O(n \log n + k^2 \log^6 n)$ time and $O(n \log n)$ space when the points are weighted.

Our objective is to improve the above result to $O(n)$ time when k is a constant. We show that we can optimally fit a k -step function to unsorted weighted points in linear time. We earlier suggested a possible approach to this problem at an OR workshop [3]. Here we flesh it out, presenting a complete and rigorous algorithm and proofs. Our algorithm exploits the well-known properties of prune-and-search along the lines in [2].

This paper is organized as follows. Section 2 introduces the notations used in the rest of this paper. It also briefly discusses how the prune-and-search technique can be used to optimally fit a 1-step function (one horizontal line) to a given set of weighted points. We then consider in Section 3 a variant of the 2-step function problem, called the anchored 2-step function problem. We discuss a “big partition” in the context of the k -partition of a point set corresponding to a k -step function in Section 4. Section 5 presents our algorithm for the optimal k -step function problem. Section 6 concludes the paper, mentioning some applications of our results.

2. Preliminaries

2.1. Model

Let $P = \{p_1, p_2, \dots, p_n\}$ be a set of n weighted points in the plane. For $1 \leq i \leq n$ let $p_i.x$ (resp. $p_i.y$) denote the x -coordinate (resp. y -coordinate) of point p_i , and let $w(p_i)$ denote its weight. The points in P are not sorted, except that $p_1.x \leq p_i.x \leq p_n.x$ holds for any $i = 1, \dots, n$.¹ Let $F_k(x)$ denote a generic k -step function, whose j^{th} segment (=step) is denoted by s_j . For $1 \leq j \leq k-1$, segment s_j represents a half-open horizontal interval $[s_j^{(l)}, s_j^{(r)})$ between two points $s_j^{(l)}$ and $s_j^{(r)}$. The last segment s_k represents a closed horizontal interval $[s_k^{(l)}, s_k^{(r)}]$. Note that $s_j^{(l)}.y = s_j^{(r)}.y$, which we denote by $s_j.y$. We assume that for any k -step function $F_k(x)$, segments s_1 and s_k satisfy $s_1^{(l)}.x = p_1.x$ and $s_k^{(r)}.x = p_n.x$, respectively. Segment s_j is said to *span* a set of points $Q \subseteq P$, if $s_j^{(l)}.x \leq p.x < s_j^{(r)}.x$ holds for each $p \in Q$. A k -step function $F_k(x)$ gives rise to a k -partition of P , $\mathcal{P} = \{P_j \mid j = 1, 2, \dots, k\}$, such that segment s_i spans

¹For the sake of simplicity we assume that no two points have the same x or y coordinate. But the results are valid if this assumption is removed.

P_i . It satisfies the *contiguity condition* in the sense that for each partition P_j , if $a.x \leq b.x$ for $a, b \in P_j$, then every point p with $a.x \leq p.x \leq b.x$ also belongs to P_j . In the rest of this paper, we consider only partitions that satisfy the contiguity condition. Fig. 1 shows an example of fitting a 4-step function $F_4(x)$.

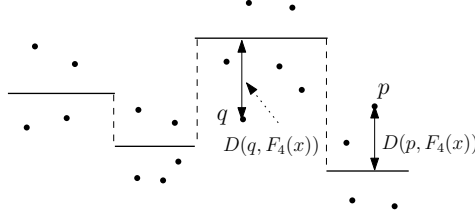


Figure 1: Fitting a 4-step function.

Given a step function $F(x)$, defined over an x -range that contains $p.x$, let $d(p, F(x))$ denote the vertical distance of p from $F(x)$. We define the *cost* of p with respect to $F(x)$ by the weighted distance

$$D(p, F(x)) \triangleq d(p, F(x))w(p). \quad (1)$$

We generalize the cost definition for a set $Q \subseteq P$ of points by

$$D(Q, F(x)) \triangleq \max_{p \in Q} \{D(p, F(x))\}. \quad (2)$$

Point p_h is said to be *critical* with respect to $F(x)$ if

$$D(p_h, F(x)) = D(P, F(x)). \quad (3)$$

Note that there can be more than one critical point with respect to a given step function. We similarly define a critical point with respect to a single segment of $F(x)$ as a point that has the maximum vertical weighted distance to it.

For a set of weighted points in the plane or on a line, the point that minimizes the maximum weighted distance to them is called the *weighted 1-center* [2]. Given a k -partition of P , $\mathcal{P} = \{P_j \mid j = 1, 2, \dots, k\}$, it is clear that $\exists P_i \in \mathcal{P}$ such that $|P_i| \geq \lfloor n/k \rfloor$. We call any such partition a *big partition*. A big partition spanned by a segment in an optimal solution plays an important role. (See Procedure **Big**(\mathcal{P}, k) in Sec. 4.3.)

2.2. Bisector

If we map each point $p_i \in P$ onto the y -axis, the *cost* of (or the weighted distance from) p_i grows linearly from 0 at $p_i.y$ in each direction as a function of y . Consider arbitrary two points p and q . Their costs intersect at either one or two points, one of which always lies between $p.y$ and $q.y$. If there are two intersections, the other intersection lies outside interval $[p.y, q.y]$ on the y -axis.

If $p.y \neq q.y$ and $w(p) = w(q)$ hold then there is only one intersection.² Let a (resp. b) be the y -coordinate of the upper (resp. lower) intersection point, where $b \leq a$. We call the horizontal line $y = a$ (resp. $y = b$) the *upper* (resp. *lower*) *bisector* of p and q . If there are only one intersection, we pretend that there were two at $b = a$, which lies between $p.y$ and $q.y$. (Note that the y -axis is shown horizontally in Figs. 2 and 3 below, where y increases to the right.)

Let \wp denote the $\lceil n/2 \rceil$ pairs, formed by pairing up the points in P arbitrarily. The cost lines of the points in each pair (p, q) intersect as shown in Figs. 2. Let

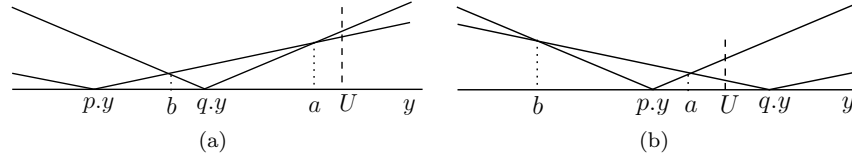


Figure 2: 1/3 of upper intersections are at $y < U$: (a) p can be ignored at $y > U$; (b) q can be ignored at $y > U$.

$y = U$ be the line at or above which at least 2/3 of the upper bisectors lie, and at or below which at least 1/3 of the upper bisectors lie. We use $\wp_{2/3}^U$ and $\wp_{1/3}^U$ to name the subsets of \wp that have these two sets of bisectors, respectively. Note that $|\wp_{2/3}^U| \geq n/2 \times 2/3 = n/3$ and $|\wp_{1/3}^U| \geq n/2 \times 1/3 = n/6$. Similarly, let $y = L$ be the line at or below which at least 2/3 of the lower bisectors lie, and at or above which at least 1/3 of the bisectors lie.³ We use $\wp_{2/3}^L$ and $\wp_{1/3}^L$ to name the subsets of \wp that have these two sets of bisectors, respectively. Note that $|\wp_{2/3}^L| \geq n/2 \times 2/3 = n/3$ and $|\wp_{1/3}^L| \geq n/2 \times 1/3 = n/6$.

Lemma 1. *We can identify $n/6$ points that can be removed without affecting the weighted 1-center for the values of their y -coordinates.*

Proof. Consider the following three possibilities.

- (i) The weighted 1-center lies above U .
- (ii) The weighted 1-center lies below L .
- (iii) The weighted 1-center lies between U and L , including U and L .

In case (i), there are two subcases, which are shown in Fig. 2(a) and (b), respectively. Since the center lies above U , we are interested in the upper envelope of the costs in the y -region given by $y > U$. In the case shown in Fig. 2(a), the costs of points p and q satisfy $d(y, p.y)w(p) < d(y, q.y)w(q)$ for $y > U$. Thus we can ignore p . In the case shown in Fig. 2(b), the costs of points p and q satisfy $d(y, p.y)w(p) > d(y, q.y)w(q)$ for $y > U$. Thus we can ignore q . Since $|\wp_{1/3}^U| \geq n/6$, in either case, one point from each such pair can be

²If $p.y = q.y$, we can ignore one of the points with the smaller weight.

³We define U and L this way, because many points could lie on them.

ignored, i.e., $1/6$ of the points in P can be eliminated, because it cannot affect the weighted 1-center. In case (ii) a symmetric argument proves that $1/6$ of the points in P can be discarded.

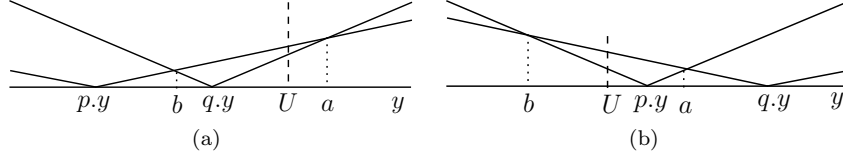


Figure 3: $2/3$ of upper bisectors are at $y > U$.

In case (iii) see Fig. 3. The costs of each pair in $\wp_{2/3}^U$ (of the $2n/3$ pairs) as functions of y intersect at most once at $y < U$. The cost functions of each pair in $\wp_{2/3}^L$ ($2n/3$ pairs) intersect at most once at $y > L$. Therefore, $\wp_{2/3}^U \cap \wp_{2/3}^L$ ($n/3$ pairs must be common to both,) i.e., both intersections of each such pair occur outside of the y -interval $[L, U]$. $|\wp_{2/3}^U \cap \wp_{2/3}^L| = n/2 \times 1/3 = n/6$. This implies that their cost functions do not intersect within in $[L, U]$, i.e., one of each pair lies above that of the other in $[L, U]$, and can be discarded. \square

2.3. Optimal 1-step function

This problem is equivalent to finding the weighted center for n points on a line. We pretend that all the points had the same x -coordinate. Then the problem becomes that of finding a weighted 1-center on a line, i.e., on the y -axis. This can be solved in linear time using Megiddo's *prune-and-search* method [2, 4, 17]. In [18] Megiddo presents a linear time algorithm in the case where the points are unweighted. For the weighted case we now present a more technical algorithm that we can apply later to solve other related problems. The following algorithm uses a parameter c which is a small integer constant.

Algorithm 1. : 1-Step(P)

1. Pair up the points of P arbitrarily.
2. For each such pair (p, q) determine their horizontal bisector lines.
3. Determine a horizontal line, $y = U$ such that $|\wp_{2/3}^U| \geq n/3$ and $|\wp_{1/3}^U| \geq n/6$ hold.
4. Determine a horizontal line, $y = L$ such that $L |\wp_{2/3}^L| \geq n/3$ and $|\wp_{1/3}^L| \geq n/6$ hold.
5. Determine the critical points for U and L .
6. If there exist critical points for U on both sides of (above and below) U , then $y = U$ defines an optimal 1-step function, $F_1^*(x)$; Stop. Otherwise, let s_U (higher or lower than U) be the side of U on which the critical point lies.
7. If there exist critical points for L on both sides of L , $y = L$ defines $F_1^*(x)$; Stop. Otherwise, let s_L (higher or lower than L) be the side of L on which the critical point lies.

8. Based on s_U and s_L , discard $1/6$ of the points from P , based on Lemma 1.
9. If the size of the reduced set P is greater than constant c , repeat this algorithm from the beginning with the reduced set P . Otherwise, determine $F_1^*(x)$ using any known method (which runs in constant time).

Lemma 2. An optimal 1-step function $F_1^*(x)$ can be found in linear time.

Proof. The recurrence relation for the running time $T(n)$ of **1-Step**(P) for general n is $T(n) \leq T(n - n/6) + O(n)$, which yields $T(n) = O(n)$. \square

3. Anchored 2-step function problem

In general, we denote an optimal k -step function by $F_k^*(x)$ and its i^{th} segment by s_i^* . Later, we need to constrain the first and/or the last step of a step function to be at a specified height. A k -step function is said to be *left-anchored* (resp. *right-anchored*), if $s_1.y$ (resp. $s_k.y$) is assigned a specified value, and is denoted by $\downarrow F_k(x)$ (resp. $F_k^\downarrow(x)$). The *anchored k -step function* problem is defined as follows. Given a set P of points and two y -values a and b , determine the optimal k -step function $\downarrow F_k^*(x)$ (resp. $F_k^{\downarrow*}(x)$) that is left-anchored (resp. right-anchored) at a (resp. b) such that cost $D(P, \downarrow F_k^*(x))$ (resp. $D(P, F_k^{\downarrow*}(x))$) is the smallest possible. If a k -step function is both left- and right-anchored, it is said to be *doubly anchored* and is denoted by $\downarrow F_k^\downarrow(x)$.

3.1. Doubly anchored 2-step function

Suppose that segment s_1 (resp. s_2) is anchored at a (resp. b). See Fig. 4(a). Let us define two functions $g(x)$ and $h(x)$ by

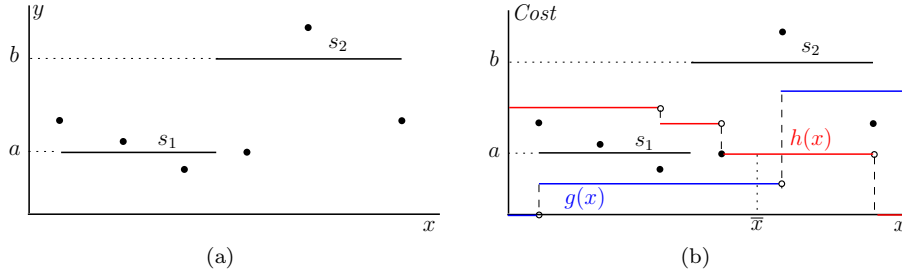


Figure 4: (a) $s_1.y = a$ and $s_2.y = b$; (b) Monotone functions $g(x)$ (in blue) and $h(x)$ (in red).

$$g(x) = \max_{p.x \leq x} \{w(p) \cdot |p.y - a| \mid p \in P\}, \quad (4)$$

$$h(x) = \max_{p.x > x} \{w(p) \cdot |p.y - b| \mid p \in P\}, \quad (5)$$

where $g(x) = 0$ for $x < p_1.x$ and $h(x) = 0$ for $x > p_n.x$. Intuitively, if we divide the points of P at x into two partitions P_1 and P_2 , then $g(x)$ (resp. $h(x)$) gives

the cost of partition P_1 (resp. P_2). See Fig. 4(b). Clearly the global cost for the entire P is minimized for any x at the lowest point in the upper envelope of $g(x)$ and $h(x)$, which is named \bar{x} . Since the points in P are not sorted, $g(x)$ and $h(x)$ are not available explicitly, but we can compute \bar{x} in linear time using the *prune-and-search* method, taking advantage of the fact that $\max\{g(x), h(x)\}$ is unimodal.

Algorithm 2. : **Doubly-Anch-2-Step**(P, a, b)

1. Initialize $P' = P$.
2. Find the point in P' that has the median x -coordinate, x_m .
3. Evaluate $g(x_m)$ (resp. $h(x_m)$) using (4) (resp. (5)).
4. If $g(x_m) = h(x_m)$ then $\bar{x} = x_m$. Stop.
5. If $g(x_m) < h(x_m)$ (resp. $g(x_m) > h(x_m)$), i.e., $\bar{x} < x_m$ (resp. $\bar{x} > x_m$), prune all the points p with $p.x < x_m$ (resp. $p.x > x_m$), from P' , remembering just the maximum cost.
6. Stop when $|P'| = 2$, and find the lowest point \bar{x} . Otherwise, go to Step 2.

We have the following lemma.

Lemma 3. *An optimal doubly anchored 2-step function can be found in linear time.*

Proof. Steps 2 and 3 of Algorithm **Doubly-Anch-2-Step**(P, a, b) can be carried out in linear time. Since Step 4 cuts the size of P' in half every time, Step 2 is entered $O(\log n)$ times. Therefore the total time is $O(n)$. \square

3.2. Left- or right-anchored 2-step function

Without loss of generality, we discuss only a left-anchored 2-step function. Given an anchor value a , we want to determine the optimal 2-step function with the constraint that $s_1^*.y = a$, denoted by $\downarrow F_2^*(x)$. See Fig. 4(a). In this case, b in (5) is not given; we need to find the optimal value for it. But assume for now that b is also given, and execute **Doubly-Anch-2-Step**(P, a, b). From the solution that it yields, can we find the direction in which to move b to find the optimal left-anchored 2-step function?

Lemma 4. *Let P_1 (resp. P_2) be the left (resp right) partition of P generated by **Doubly-Anch-2-Step**(P, a, b) such that $s_1.y = a$ (resp. $s_2.y = b$), where $a < b$ without loss of generality. Assume that P_1 is maximal in the sense that the boundary between P_1 and P_2 cannot be moved to the right without increasing the cost of the solution.*

- (a) *If $D(P_1, s_1) \geq D(P_2, s_2)$ then the optimal right-anchor cannot lie above $y = b$.*
- (b) *If $D(P_1, s_1) < D(P_2, s_2)$ and there is a critical point in P_2 for $s_2.y = b$ above $y = b$, then the optimal right-anchor cannot lie below $y = b$.*
- (c) *If $D(P_1, s_1) < D(P_2, s_2)$ and there is a critical point in P_2 for $s_2.y = b$ below $y = b$, then the optimal right-anchor cannot lie above $y = b$.*

Proof. (a) Assume first that the critical point p_1 for s_1 lies below s_1 ($y = a$). Then we cannot reduce the cost by changing the value of b . Therefore, assume that p_1 lies above s_1 . By the definition of $\{P_1, P_2\}$, the leftmost point in P_2 lies above $y = b$, and moving the boundary between P_1 and P_2 to the right increases the cost, which is due to the weighted distance between s_1 and the new point in P_1 , and this increase is independent of the value of b . Moving this boundary to the left cannot decrease the cost, until p_1 becomes a part of P_2 , and even then a decrease is not possible unless b is made smaller. Otherwise, $\{P_1, P_2\}$ wouldn't be optimal with the current b .

(b) We know that moving the boundary between P_1 and P_2 to the right increases the cost if b is kept at the same value. The cost increases if b is made smaller.

(c) Symmetric to Case (b). □

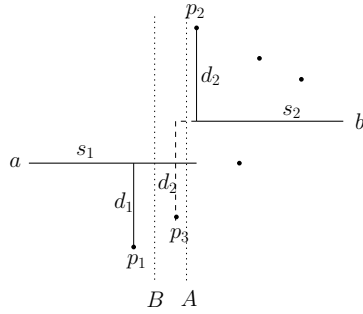


Figure 5: An example for $l\text{-Anch-2-Step}(P, a)$.

Example 1. In Fig. 5, assume that d_2 is slightly larger than d_1 . We have a doubly anchored solution with the minimum cost (weighted distance) equal to d_2 . When the boundary between P_1 and P_2 is at A , we can reduce the cost of the optimal solution by moving b up. We cannot do so if the boundary is at B , because $D(p_3, s_2)$ would increase. This is why we maximize P_1 in Step 5(b) of Algorithm 3 presented below. □

To make use of the prune-and-search method, we want to find the big partition (defined in Sec. 2.1), P_1 or P_2 , that is spanned by one segment of ${}^{\downarrow}F_2^*(x)$. If P_1 is the big partition, we can eliminate all the points belonging to it, without affecting ${}^{\downarrow}F_2^*(x)$ that we will find. See Step 4 of the Algorithm 3 given below. We then repeat the process with the reduced set P . If P_2 is the big partition, on the other hand, we need to do more work, similar to what we did to find an optimal 1-step function. Namely, we determine values U and L for P_2 by executing Algorithm 1-Step(P_2). We then find a doubly anchored 2-step solution for P with left anchor a and right anchor U .

Algorithm 3. : $l\text{-Anch-2-Step}(P, a)$

1. Divide P into left partition P_1 and right partition P_2 , whose sizes differ by at most one.⁴
2. Let s_1 be the segment with $s_1.y = a$ spanning P_1 , and let s_2 be the 1-step (optimal) solution for P_2 .⁵
3. If $D(P_1, s_1) = D(P_2, s_2)$ then output $\{s_1, s_2\}$, which defines $\downarrow F_2^*(x)$. Stop.
4. If $D(P_1, s_1) < D(P_2, s_2)$, remove from P the points of P_1 , except the critical point for s_1 . Go to Step 6.
5. If $D(P_1, s_1) > D(P_2, s_2)$ then carry out the following steps.
 - (a) Determine points U and L for P_2 as described in Algorithm 1-Step(P).
 - (b) Execute Doubly-Anch-2-Step(P, a, U), and find the solution whose left partition is maximal. Repeat it with right anchor L .
 - (c) Eliminate $1/6$ of the points of P_2 from P , based on the two solutions (as in Steps 6–8 of Algorithm 1-Step(P)).
6. If $|P| > c$ (a small constant), repeat Steps 1 to 4. Otherwise, optimally solve the problem in constant time, using a known method.

In the example in Fig. 5, assume that b is not given, and s_2 is determined by Step 2. Then we have $D(P_1, s_1) > D(P_2, s_2)$, and Step 5 applies. According to Step 5(a), we determine U . We then find the doubly anchored solution with the right anchor set to $b = U$.

Lemma 5. *Algorithm l-Anch-2-Step(P, a) computes $\downarrow F_2^*(x)$ correctly, and runs in linear time.*

Proof. Step 3 is obviously correct. If $D(P_1, s_1) < D(P_2, s_2)$ holds in Step 4, then the first partition of $\downarrow F_2^*(x)$ contains P_1 . We need to keep the critical point for a , but all other points of P_1 can be ignored from now on because P_1 will expand. If $D(P_1, s_1) > D(P_2, s_2)$ holds in Step 5, then the first partition of $\downarrow F_2^*(x)$ is contained in P_1 .

Each iteration of Steps 3 and 4 will eliminate at least $1/2 \times 1/6 = 1/12$ of the points of P . Such an iteration takes linear time in the input size. The total time needed for all the iterations is therefore linear. \square

4. k -step function

4.1. Approach

To design a recursive algorithm, assume that for any set of points $Q \subset P$, we can find the optimal $(j - 1)$ -step function and the optimal left- and right-anchored j -step function for any $2 \leq j < k$ in $O(|Q|)$ time, where k is a constant. We have shown that this is true for $k = 2$ in the previous two sections. So the basis of induction holds.

Given an optimal k -step function $F_k^*(x)$, for each i ($1 \leq i \leq k$), let P_i^* be the set of points vertically closest to segment s_i^* . By definition, the partition

⁴As before, we assume that the points have different y -coordinates.

⁵Segment s_2 can be found in $O(|P_2|)$ time by Lemma 2.

$\{P_i^* \mid i = 1, 2, \dots, k\}$ satisfies the contiguity condition. It is easy to see that for each segment s_i^* , there are (local) critical points with respect to s_i^* , lying on the opposite sides of s_i^* .

In finding an optimal k -step function, we first identify a big partition that will be spanned by a segment in an optimal solution. By Lemma 7, such a big partition always exists. Our objective is to eliminate a constant fraction of the points in a big partition. This will guarantee that a constant fraction of the input set is eliminated when k is a fixed constant. The points in the big partition other than two critical points are “useless” and can be eliminated from further considerations.⁶ This elimination process is repeated until the problem size gets small enough to be solved by an exhaustive method in constant time.

4.2. Feasibility test

Given a weighted distance (=cost) D , a point set P is said to be D -feasible if there exists a k -step function $F_k(x)$ such that $D(P, F_k(x)) \leq D$. To test D -feasibility we first try to identify the first segment s_1 of a possible k -step function $F_k(x)$. To this end we compute the median m of $\{p_i.x \mid i = 1, 2, \dots, n\}$ in $O(n)$ time, and divide P into two parts $P_1 = \{p_i \mid p_i.x \leq m\}$ and $P_2 = \{p_i \mid p_i.x > m\}$, which also takes $O(n)$ time. Note that $|P_1| \leq \lceil |P|/2 \rceil$ and $|P_2| \leq \lceil |P|/2 \rceil$ hold. We then find the intersection I of the y -intervals in $\{|p_i.y - y| \leq D \mid p_i \in P_1\}$. Assuming that P is D -feasible, then we have two cases.

Case (a): $[I] = \emptyset$ s_1 ends at some point $p_j \in P_1$. Throw away all the points in P_2 and look for the longest s_1 limited by cost D , considering only the points in P_1 from the left.

Case (b): $[I] \neq \emptyset$ s_1 may end at some point $p_j \in P_2$. Throw away all the points in P_1 and look for the longest s_1 , using I and the points in P_2 from the left.

Clearly, we can find the longest s_1 in $O(n)$ time. Remove the points spanned by s_1 from P , and find s_2 in $O(n)$ time, and so on. Since we are done after finding k steps $\{s_1, \dots, s_k\}$, it takes $O(kn)$ time.

Lemma 6. *We can test D -feasibility in $O(kn)$ time.* □

4.3. Identifying a big partition

Lemma 7. *Let $\mathcal{P} = \{P_i \mid i = 1, \dots, k\}$ be any k -partition of P , satisfying the contiguity condition, such that the sizes of the partitions differ by no more than 1, and let $\{P_i^* \mid i = 1, \dots, k\}$ be an optimal k -partition. Then there exists an index j such that P_j is a big partition spanned by s_j^* .*

Proof. Let j be the smallest index such that $s_j^{(r)}.x \leq s_j^{*(r)}.x$. Such an index must exist, because if $s_j^{(r)}.x > s_j^{*(r)}.x$ for all $1 \leq j \leq k-1$ then $s_k^{(r)}.x = s_j^{*(r)}.x$. We clearly have $s_j \subset s_j^*$, which implies that s_j^* spans P_j . □

⁶Note that there may be more than two critical points in which case all but two are “useless.”

Given a point set P in the x - y plane, let $\mathcal{P} = \{P_i \mid i = 1, \dots, k\}$ be any k -partition of P , satisfying the contiguity condition, such that the sizes of the partitions differ by no more than 1. The following procedure returns a big partition P_j spanned by s_j^* , whose existence was proved by Lemma 7. Since $P = \cup\{P_i \mid P_i \in \mathcal{P}\}$, P is implicit in the input to the next procedure.

Procedure 1. : $\text{Big}(\mathcal{P}, k)$

1. Using Algorithm 1-Step(P), compute the optimal 1-step function for P_1 and let D_1 be its cost for P_1 . If P is not D_1 -feasible (i.e., $D(P, F_k^*(x)) > D_1$), then return P_1 and stop.⁷
2. Using Algorithm 1-Step(P), compute the optimal 1-step function for P_k and let D'_k be its cost for P_k . If P is not D'_k -feasible (i.e., $D(P, F_k^*(x)) > D'_k$), then return P_k and stop.
3. Find an index j ($1 < j < k$) such that for $D_{j-1} = D(\cup_{i=1}^{j-1} P_i, F_{j-1}^*(x))$ P is D_{j-1} -feasible, and for $D_j = D(\cup_{i=1}^j P_i, F_j^*(x))$ P is not D_j -feasible.⁸ Return P_j and stop.

Lemma 8. Procedure $\text{Big}(\mathcal{P}, k)$ is correct.

Proof. It is clear that Steps 1 and 2 are correct. To show that Step 3 is also correct, we *stretch* a step s of an optimal step function by making it as long as possible as follows. Move $s^{(l)}.x$ (resp. $s^{(r)}.x$) to the left (resp. right) as far as possible without changing the cost of the step function. The step that has been stretched is called a *stretched step*. Let us assume without loss of generality that s_j^* corresponding to P_j returned by Step 3 is stretched. Since $D_{j-1} \geq D^*$, we must have $s_j^{*(l)}.x \leq s_j^{(l)}.x$.

The optimal solution $F_j^*(x)$ for $\cup_{i=1}^j P_i$ has cost D_j , which is too small for P to be D_j -feasible. Regarding the remaining points $\cup_{i=j+1}^k P_i$, let $G_j^*(x)$ denote the optimal $(k-j)$ -step function for this point set. If $D(\cup_{i=j+1}^k P_i, G_j^*(x)) \leq D_j$, the P would be D_j -feasible. Since it is not, $s_j^{(r)}.x$ would be stretched to the right under the optimal solution $F_k^*(x)$, i.e., $s_j^{*(r)}.x \geq s_j^{(r)}.x$. Together with $s_j^{*(l)}.x \leq s_j^{(l)}.x$, it follows that P_j is spanned by s_j^* . \square

Lemma 9. Procedure $\text{Big}(\mathcal{P}, k)$ runs in linear time in n .

Proof. In Step 1, the optimal 1-step function for P_1 can be found in $O(|P_1|)$ time by Lemma 2, and it takes $O(kn)$ time to test if P is not D_1 -feasible by Lemma 6. Similarly, Step 2 can be carried out in $O(n)$ time. To carry out Step 3, we compute, using binary search, $\lceil \log n \rceil$ values out of $\{D_i \mid 1 \leq i \leq k-1\}$, which takes $O(f(k)n)$ time for some function $f(k)$, under the assumption that any i -step function problem, $i < k$, is solvable in time linear in the size of the input point set. \square

⁷There exists an optimal solution for P in which s_1^* spans P_1 .

⁸This means that $D_{j-1} \geq D^*$ and $D_j < D^*$, where D^* is the cost of the optimal solution for P . Unless $P_i^* = P_i$ for all i , such an index j always exists. [We should indicate why.]

5. Algorithm

5.1. Optimal k -step function

In this section we are assuming that we can solve any $(j - 1)$ -step and anchored j -step function problems for any $2 \leq j < k$. We have shown that this is true for $k = 2$ in the previous section. So the basis of recursion holds.

Let us find an optimal doubly anchored k -step function, $\downarrow F_k^{\downarrow*}(x)$, which consists of k horizontal segments, $s_i^*, i = 1, 2, \dots, k$, satisfying $s_1^{*(l)}.x = p_1.x$, $s_1^*.y = a$, $s_k^{*(r)}.x = p_n.x$, and $s_k.y = b$, where a and b are given constants. Let P_i^* be the set of points of P vertically closest to s_i^* . For each segment s_i^* , there are critical points with respect to s_i^* , lying on the opposite sides of s_i^* . In order to find $\downarrow F_k^{\downarrow*}(x)$, we first execute **Big**(\mathcal{P}, k) and identify a big partition containing at least $\lfloor n/k \rfloor$ points, which are vertically closest to the same segment in some optimal solution.

Once a big partition, say P_j , is identified, We first determine U and L for P_j as described in Algorithm **1-Step**(P). To illustrate the idea, let us consider a special case where $j = 1$ and $k = 2$. We execute **l -Anch-2-Step**(P, U) and **l -Anch-2-Step**(P, L) and determine s_U and s_L as in Algorithm **1-Step**(P). We can thus eliminate $1/6$ of the points in P_1 . We repeat this with the reduced P . It may turn out that the right partition is the big partition in the next round. Then we can repeat the above process symmetrically. Eventually, the size of P gets small enough, so that we can find the solution using an exhaustive method.

For a general k and $j > 1$, we need to find the left- and right-anchored solution for U and L , and prune $1/6$ of the points in P_j using **Prune-Big**(k, P_j), given below, which is very similar to Algorithm **1-Step**(P). Let P_j be a big partition spanned by s_j^* , which is an input to the following procedure.

Procedure 2. : **Prune-Big**(k, P_j)

Output: $1/6$ of points in P_j removed.

1. Determine U and L for P_j as in Algorithm **1-Step**(P).
2. If $j > 1$, find two right-anchored j -step functions $F_j^{\downarrow*}(x)$ for $\cup_{i=1}^j P_i$, one anchored by L and the other anchored by U .
3. If $j < k$, find two left-anchored $(k - j + 1)$ -step functions $\downarrow F_{k-j+1}^*(x)$ for $\cup_{i=j}^k P_i$, one anchored by L and the other anchored by U .
4. Identify $1/6$ of the points in P_j with respect to L and U , which are “useless”⁹ based on $F_j^{\downarrow*}(x)$ and $\downarrow F_{k-j+1}^*(x)$ found above, and remove them from P .

Lemma 10. **Prune-Big**(k, P_j) runs in linear time when k is a constant.. \square

We can now describe our algorithm formally as follows.

⁹See Step 8 of Algorithm **1-Step**(P).

Algorithm 4. : $k\text{-Step}(P)$.

Output: Optimal k -step function $F_k^*(x)$

1. Divide P into partitions $\{P_i \mid i = 1, 2, \dots, k\}$, satisfying the contiguous condition, such that their sizes differ by no more than one.
2. Execute Procedure $\text{Big}(P, k)$ to find a big partition P_j spanned by s_j^* .
3. Execute Procedure $\text{Prune-Big}(k, P_j)$.
4. If $|P| > c$ for some fixed c , repeat Steps 1 to 3 with the reduced P .

5.2. Analysis of algorithm

To carry out Step 1 of Algorithm $k\text{-Step}(P)$, we first find the $(hn/k)^{th}$ smallest among $\{p_i.x \mid 1 \leq i \leq n\}$, for $h = 1, 2, \dots, k-1$. We then place each point in P into k partitions delineated by these $k-1$ values. It is clear that this can be done in $O(kn)$ time.¹⁰ As for Step 2, we showed in Sec. 4.3 that finding a big partition spanned by an optimal step s_j^* takes $O(n)$ time, since k is a constant. Step 3 also runs in $O(n)$ time by Lemma 10. Since Steps 1 to 3 are repeated $O(\log n)$ times, each time with a point set whose size is at most a constant fraction of the size of the previous set, the total time is also $O(n)$, when k is a constant. By solving a recurrence relation for the running time of Algorithm $k\text{-Step}(P)$, we can show that it runs in $O(2^{2k \log k} n) = O(k^{2k} n)$ time.

Theorem 1. *Given a set of n points in the plane $P = \{p_1, p_2, \dots, p_n\}$, we can find the optimal k -step function that minimizes the maximum distance to the n points in $O(k^{2k} n)$ time.* \square

Thus the algorithm is optimal for a fixed k .

6. Conclusion and Discussion

We have presented a linear time algorithm to solve the optimal k -step function problem, when k a constant. Most of the effort is spent on identifying a “big partition.” It is desirable to reduce the constant of proportionality.

The *size- k histogram construction problem* [13], where the points are not weighted, is similar to the problem we addressed in this paper. Its generalized version, where the points are weighted, is equivalent to our problem, and thus can be solved in optimal linear time when k is a constant. The *line-constrained k center problem* is defined by: Given a set P of weighted points in the plane and a horizontal line L , determine k centers on L such that the maximum weighted distance of the points to their closest centers is minimized. This problem was solved in optimal $O(n \log n)$ time for arbitrary k even if the points are sorted [14, 20]. Our algorithm presented here can be applied to solve this problem in $O(n)$ time if k is a constant.

¹⁰This could be done in $O(n \log k)$ time.

A possible extension of our work reported here is to use a cost other than the weighted vertical distance. There is a nice discussion in [13] on the various measures one can use. Our complexity results are valid if the cost is more general than (1), in particular, $D(p, F(x)) \triangleq d(p, F(x))^2 w(p)$, which is often used as an error measure.

Acknowledgement

This work was supported in part by Discovery Grant #13883 from the Natural Science and Engineering Research Council (NSERC) of Canada and in part by MITACS, both awarded to Bhattacharya.

Reference

References

- [1] Ajtai, M., Komlós, J., Szemerédi, E.: An $O(n \log n)$ sorting network. In: Proc. 15th Annual ACM Symp. Theory of Computing (STOC). pp. 1–9 (1983)
- [2] Bhattacharya, B., Shi, Q.: Optimal algorithms for the weighted p -center problems on the real line for small p . In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 529–540. Springer, Heidelberg (2007)
- [3] Bhattacharya, B., Das, S.: Prune-and-search technique in facility location. In: Proc. 55th Conf. Canadian Operational Research Society (CORS). p. 76 (May 2013)
- [4] Chen, D.Z., Li, J., Wang, H.: Efficient algorithms for the one-dimensional k -center problem. Theoretical Comp. Sci. 592, 135–142 (August 2015)
- [5] Chen, D.Z., Wang, H.: Approximating points by a piecewise linear function: I. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 224–233. Springer, Heidelberg (2009)
- [6] Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. J. ACM 34, 200–208 (1987)
- [7] Díaz-Báñez, J., Mesa, J.: Fitting rectilinear polygonal curves to a set of points in the plane. European J. Operations Research 130, 214–222 (2001)
- [8] Fournier, H., Vigneron, A.: Fitting a step function to a point set. Algorithmica 60, 95–101 (2011)
- [9] Fournier, H., Vigneron, A.: A deterministic algorithm for fitting a step function to a weighted point-set. Information Processing Letters 113, 51–54 (2013)

- [10] Frederickson, G.: Optimal algorithms for tree partitioning. In: Proc. 2nd ACM-SIAM Symp. Discrete Algorithms. pp. 168–177 (1991)
- [11] Frederickson, G., Johnson, D.: Generalized selection and ranking. SIAM J. Computing 13(1), 14–30 (1984)
- [12] Gabow, H., Bentley, J., Tarjan, R.: Scaling and related techniques for geometry problems. In: Proc. 16th Annual ACM Symp. Theory of Computing (STOC). pp. 135–143 (1984)
- [13] Guha, S., Shim, K.: A note on linear time algorithms for maximum error histograms. IEEE Trans. Knowl. Data Eng. 19, 993–997 (2007)
- [14] Karmakar, A., Das, S., Nandy, S.C., Bhattacharya, B.: Some variations on constrained minimum enclosing circle problem. J. Comb. Opt. 25(2), 176–190 (2013)
- [15] Liu, J.Y.: A randomized algorithm for weighted approximation of points by a step function. In: Proc. 4th Ann. Int. Conf. Combinatorial Optimization and Applications (COCOAA), Springer-Verlag. vol. LNCS 6509, pp. 300–308 (2010)
- [16] Lopez, M., Mayster, Y.: Weighted rectilinear approximation of points in the plane. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 642–653. Springer, Heidelberg (2008)
- [17] Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. J. ACM 30, 852–865 (1983)
- [18] Megiddo, N.: Linear-time algorithms for linear-programming in R^3 and related problems. SIAM J. Computing 12, 759–776 (1983)
- [19] Wang, D.: A new algorithm for fitting a rectilinear x -monotone curve to a set of points in the plane. Pattern Recognition Letters 23, 329–334 (2002)
- [20] Wang, H., Zhang, J.: Line-constrained k -median, k -means and k -center problems in the plane. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 3–14. Springer, Heidelberg (2014)